Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 1 (2010) 97–106
Procedia Computer Science
www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2010

Scaling of ab-initio nuclear physics calculations on multicore computer architectures

Pieter Maris[a,*], Masha Sosonkina[b], James P. Vary[a], Esmond Ng[c], Chao Yang[c]

[a] *Department of Physics, Iowa State University, Ames, IA 50011, USA*
[b] *Scalable Computing Laboratory, Ames Laboratory, Iowa State University, Ames IA 50011, USA*
[c] *Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA*

Abstract

We discuss the scaling behavior of a state-of-the-art Configuration Interaction code for nuclear physics on modern multicore computer architectures. In the CI approach, the quantum many-body problem is expressed as a large sparse symmetric eigenvalue problem, of which the lowest eigenvalues and eigenvectors have to be computed. We compare the performance of the pure MPI version with the hybrid MPI/OpenMP code on Cray XT4 and XT5 platforms. For large core counts (typically 5,000 and above), the hybrid version is more efficient than pure MPI.

Keywords: hybrid MPI/OpenMP, scaling, MFDn

1. Introduction

The structure of the atomic nucleus and its interactions with matter and radiation have long been the foci of intense theoretical research aimed at a quantitative understanding based on the underlying strong inter-nucleon potentials. Once validated, a successful approach promises predictive power for key properties of short-lived nuclei that are present in stellar interiors and in other nuclear astrophysical settings. Moreover, new medical diagnostic and therapeutic applications may emerge as exotic nuclei are predicted and produced in the laboratory. Fine tuning nuclear reactor designs to reduce cost and increase both safety and efficiency are also possible outcomes with a high precision theory.

Solving for nuclear properties with the best available nucleon-nucleon (NN) potentials [1, 2, 3], supplemented by 3-body (NNN) potentials as needed [4, 5, 6, 7], using a quantum many-particle framework that respects all known symmetries of the potentials is referred to as an "*ab initio*" problem and is recognized to be computationally hard. To date, there are three *ab initio* approaches that have proven applicable to nuclei with more than 6 nucleons: Green's Function Monte Carlo [5], Coupled Cluster [8] and No Core Configuration Interaction (CI) methods [7, 9, 10, 11, 12]. All three are potentially exact methods of solution, and are computationally intensive methods funded under the SciDAC program at the U.S. Department of Energy. Here we discuss the scaling behavior of MFDn (Many Fermion Dynamics for nuclear structure) [13, 14, 15], which is a state-of-the-art numerical code for *ab initio* CI calculations

*Corresponding author
Email address: pmaris@iastate.edu (Pieter Maris)

1877-0509 © 2010 Published by Elsevier Ltd.
doi:10.1016/j.procs.2010.04.012

of light nuclei. MFDn has good scaling properties using MPI on existing supercomputing architectures and recent algorithmic improvements have significantly improved its overall performance [15]. Here, we demonstrate how the improved MFDn code may be adapted to multicore platforms.

Processors equipped with 4 or more cores became ubiquitous in modern high-performance computers. The number of cores per node is growing further to keep up with Moore's law bringing the total number of cores in a supercomputer to tens of thousands. To sustain the overall performance growth, one must prevent the cores from idling due to the "memory wall", which appears because the memory speed and access do not match the growth of processing speed [16]. A way to deal with this disparage is to mask memory accesses and communications with computations. Thus, algorithms exhibiting very high degrees of parallelism and data locality are being developed for multicore architectures. A concomitant approach is to extract additional parallelism using *hybrid* message passing and shared memory programming paradigms. In particular, Message Passing Interface (MPI) [17] and Open MP [18] may be used to communicate among inter-node cores and intra-node cores, respectively. Recent studies [19, 20] showed a feasibility of this approach.

In this paper, a hybrid MPI/OpenMP approach is applied to MFDn, which makes the package significantly more efficient than when a pure MPI implementation is used. We show experiments on Cray XT5 and XT4 architectures at the U.S. Department of Energy Oak Ridge National Laboratory (ORNL) and National Energy Research Supercomputing Center (NERSC), respectively.

2. No-Core Full Configuration calculations for nuclear structure

The nuclear wave functions satisfy the Schrödinger equation, that is, they are eigenfunctions of the Hamiltonian operator H, with the eigenvalues being the corresponding energy

$$H\,|\psi_i\rangle \quad = \quad E\,|\psi_i\rangle\,. \tag{1}$$

In the No-Core Shell Model (NCSM) [9] and No-Core Full Configuration (NCFC) [10, 11, 12] approaches, the wave function Φ of a nucleus consisting of A nucleons (protons and neutrons) is expanded in an A-body basis of Slater Determinants of single-particle states; that is

$$\Phi(r_1,\ldots,r_A) \;:=\; \langle r_1,\ldots,r_A|\psi\rangle \quad = \quad \sum c_k \Psi_k(r_1,\ldots,r_A)\,, \tag{2}$$

with Ψ_k the A-body basis functions. The interaction, typically a two-body (NN), sometimes supplemented by a three-body (NNN) potential, is also expressed in this basis, as is the kinetic energy. Conventionally, one uses a harmonic oscillator basis for the single-particle states, but it is straightforward to extend this approach to a more general single-particle basis.

In principle, this basis is an infinite-dimensional complete basis. For practical calculations one has to truncate the basis to a finite dimension. In the NCSM this truncation is governed by $N_{\max}$, the maximum number of harmonic oscillator excitation quanta in the many-body basis. In the truncated basis space, the Schrödinger equation becomes a finite matrix equation with a real, symmetric, sparse matrix. The lowest eigenvalue of this matrix corresponds to the ground state energy, but typically we are also interested in the (low-lying) spectrum; we therefore typically solve for the 10 to 15 lowest eigenvalues using the Lanczos algorithm. By performing calculations in a series of bases of increasing dimension, one can extrapolate physical observables such as the mass or binding energy of a nucleus to the result in the full, infinite-dimensional basis, with numerical error bars (NCFC approach) [10, 11, 12].

In order to extrapolate to the infinite-dimensional basis, we need a series of calculations with increasing $N_{\max}$, and this is where the computational challenge is. As $N_{\max}$ increases, the dimension of the many-body basis increases exponentially, see Fig. 1. Furthermore, the sparsity of the matrix depends on the rank K of the potential: a two-body potential ($K = 2$) leads to a much sparser many-body matrix than a three-body potential ($K = 3$). As the matrix size increases, there is a clear need for high-performance computing: for a reliable extrapolation to the infinite model-space, we would like to have $N_{\max} \geq 10$. On currently available machines, the largest many-body basis dimension that MFDn can deal with for three-body forces is about 1 billion, and for two-body forces we can go to about 10 to 15 billion.

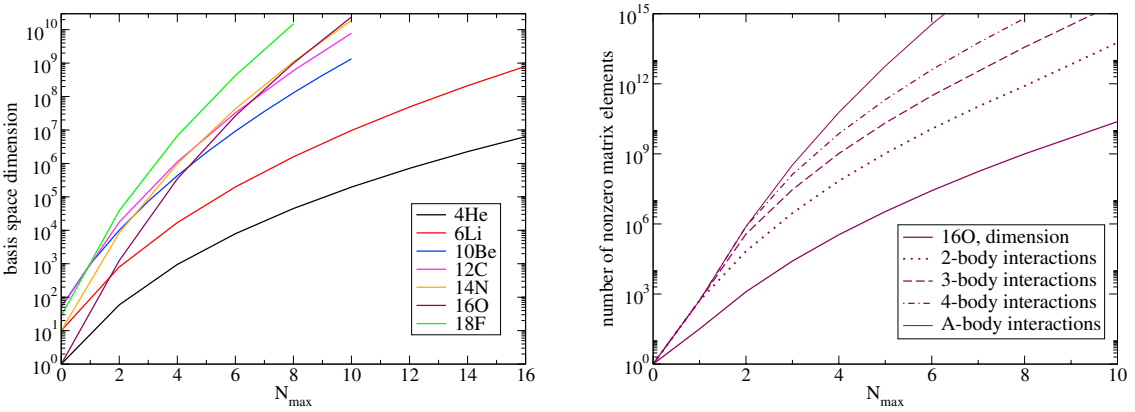

Figure 1: Left: basis space dimension as function of $N_{\max}$ for a range of light nuclei; Right: number of non-zero matrix elements in the many-body matrix as as function of $N_{\max}$ for two-body, three-body, and four-body interactions.

3. Brief description of the code MFDn

The code MFDn, which is a parallel code for nuclear structure calculations using F90 and MPI, has been in development for almost two decades. In the early development [13, 14] of the code the main focus has been efficient use of memory; significant improvements in its performance have been made over the last 3 years [15, 21, 22, 23] under the US Department of Energy's Scientific Discovery through Advanced Computing (SciDAC) Program. Before presenting its scaling properties, we first briefly outline the structure of the code.

MFDn constructs the many-body basis states and the Hamiltonian matrix, and solves for the lowest eigenstates using the Lanczos algorithm. At the end of a run, it writes the nuclear wave functions to file, and evaluates selected physical observables which can be compared to experimental data (the wave functions themselves are not experimentally observable). The matrix is distributed in a 2-dimensional fashion over the processors, as indicated in Fig. 2, and we work with the lower triangle only, because the matrix is symmetric (and real). The Lanczos vectors, needed for re-orthoganalization after every matrix-vector multiplication, are stored distributed over all processors. Because of the 2-dimensional distribution of the matrix, it runs on $n(n+1)/2$ processors, where n is the number of "diagonal" processors (see Fig. 2).

There are several distinct phases in the code, each with its own scaling and performance characteristics.

- Setup of basis
  The first phase is a setup phase, in which the single-particle basis states are defined, and the A-body basis is developed. The construction of the A-body basis is done on the n diagonal processors only, so this part does not scale very well; however, generally this is only a very small part of the total runtime. The basis states are constructed in a lexicographical order and distributed in a round-robin fashion [13, 15].

- Construction of sparse, real, symmetric matrix
  A matrix element H_{ij} can only be nonzero if the two A-body basis states i and j differ by at most K single-particle states (typically, we have $K = 2$ or $K = 3$). With a lexicographical ordering of the A-body basis states on a single processor, neighboring basis states often differ by only one single-particle state, and nonzero matrix elements tend to occur in blocks, see Fig. 2. A naive 2-dimensional distribution of this matrix leads to poor load-balancing, both in terms of memory needs and in terms of CPU time. With the round-robin distribution of the A-body basis states over the n diagonal processors, neigboring basis states tend to have very few single-particle states in common and the nonzero matrix elements appear to be randomly distributed (right panel of Fig. 2). This distribution of the matrix leads to excellent load-balancing, and may be useful for other large sparse matrix problems.

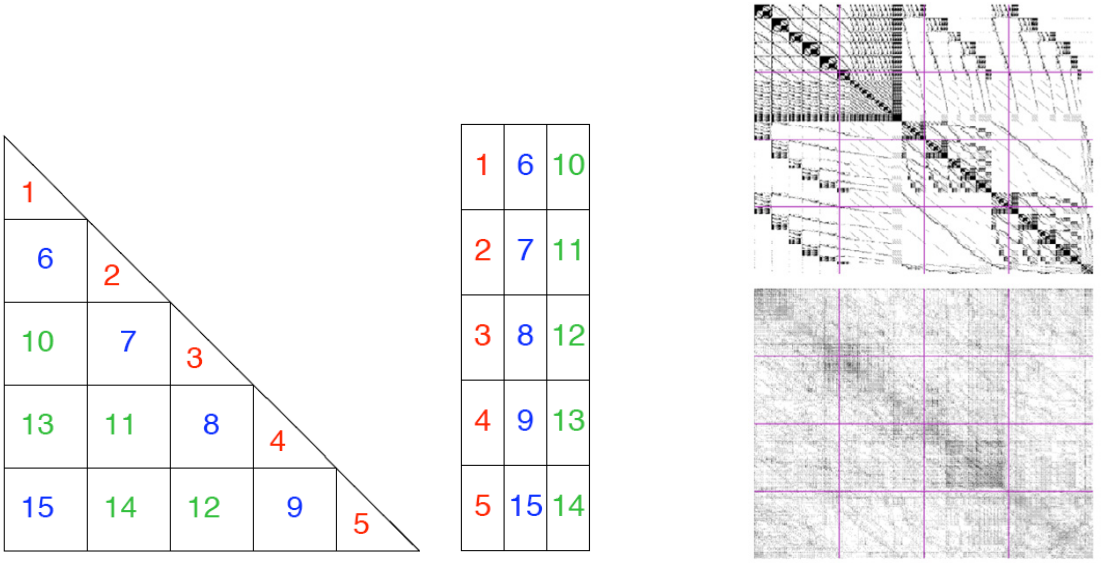

Figure 2: Left: Two-dimensional distribution of the lower triangle of the matrix and of the Lanczos vectors over $n(n+1)/2$ processors with $n = 5$. The first n processors (in red) are referred to as the "diagonal" processors. Right: load-balanced distribution of nonzero matrix elements on $n(n+1)/2$ processors with $n = 4$.

The disadvantage of the round-robin distribution of the basis states is that the construction of the matrix becomes much more cumbersome and time-consuming as the matrix size increases. In order to deal efficiently with the problem of determining which matrix elements can be nonzero, a 'multilevel blocking scheme' was introduced, based on groups of single-particles states [15]. The performance of the code does depend on the particular choice of how to create these groups – the most efficient choice depends not only on the nucleus, but also on the number of processors. For simplicity, we have kept these groups constant in the scaling experiments presented below. Once we know which matrix elements are nonzero, we calculate these matrix elements, and store the sparse matrix in compressed sparse column (CSC) format. There is almost no MPI communication in this part of the code, so it should scale reasonably well, provided that the work is well load-balanced.

- Lanczos iterations
  Once we have constructed the matrix (and stored in memory, using CSC format, distributed over all processors), we solve for the lowest eigenvalues using an iterative Lanczos algorithm. Each iteration consists of a matrix-vector multiplication, followed by an orthogonalization against all previous Lanczos vectors (which are also all stored in memory, distributed over all processors). Due to the 2-dimensional distribution of the matrix, there is a significant amount of MPI-communication in this phase: After each matrix-vector multiplication, the resulting output vector needs to be accumulated on the n diagonal processors. Next, this vector needs to be distributed to all processors in order to do the orthogonalization, and finally the new input vector needs to be distributed to all processors. After a fixed number of Lanczos iterations (which is an input variable), the lowest eigenvalues and the corresponding wave functions are written to disk from the n diagonal processors. This can be done either with MPI-IO, or with HFD5 [23]; in the scaling experiments presented below we used MPI-IO.

- Evaluation of observables
  Finally, we read the wave function back in, and evaluate physical observables such as the rms radius of the nucleus, its dipole and quadrupole moments, and radiative transitions between the ground state and excited states. This part of the code is similar to the calculation of the matrix elements. It does not involve a lot of MPI communication, and should therefore scale reasonably well, provided that the code is well load-balanced.

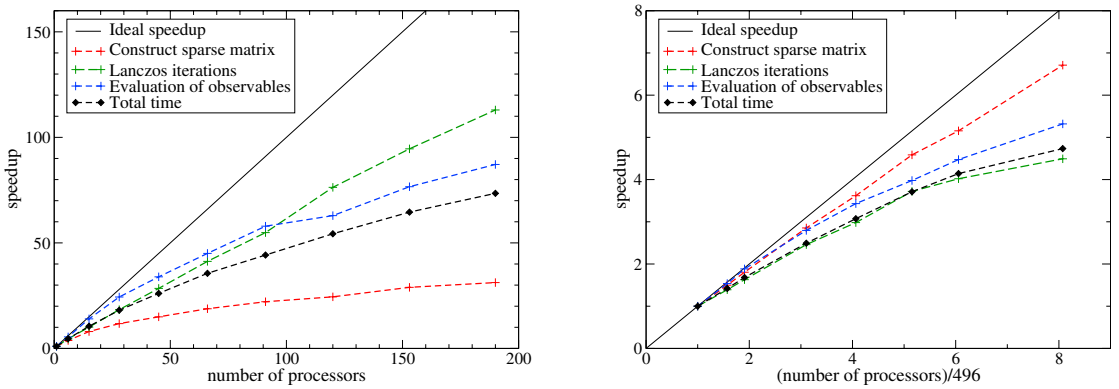

Figure 3: Speedup for ^{12}C at $N_{\max} = 4$ (left) and for ^{6}Li at $N_{\max} = 12$ (right) on Franklin (Cray XT4 at NERSC with 8 GB quad-core processors), using NN forces only. (The ^{6}Li, $N_{\max} = 12$ runs need at least 496 processors with 2 GB of memory each.)

4. Scalibility of MFDn

The biggest challenge for our calculations is the amount of memory required to keep the sparse matrix and the Lanczos vectors all in memory. We therefore get significantly better performance on the Cray XT4 and XT5 platforms than on the BlueGene P. Here we report the results of several scaling experiments on Franklin at NERSC and on Jaguar at ORNL. Franklin is a Cray XT4 at NERSC with 9,572 compute nodes. Each node is a 2.3 GHz quad-core AMD Opteron processor (Budapest), with 8 GB of memory (2 GB per core). The compute nodes are connected through a SeaStar2 router.

In addition to our scaling experiments on Franklin at NERSC, we performed scaling runs on the Cray XT5 partition of Jaguar at ORNL. This Cray XT5 system has 18,688 compute nodes connected through a SeaStar2+ router. Since October 2009, each compute node contains two 2.6 GHz hex-core AMD Opteron processors (Istanbul) with 16 GB (8 GB per processor, 1.33 GB per core). With 12 cores per compute node, this system allows in principle for better testing of the scalibility of the hybrid MPI/OpenMP code.

We restrict ourselves to strong scaling, and considered several problem sizes, as indicated in Table 1.

nucleus	model space	dimension	# of nonzeros
^{12}C	$N_{\max} = 4$	1,118,926	279,405,126
^{6}Li	$N_{\max} = 12$	48,887,656	73,247,559,498
^{14}N	$N_{\max} = 8$	1,090,393,922	900,538,668,203
^{14}Be	$N_{\max} = 8$	2,790,412,009	2,769,980,757,614
^{13}C	$N_{\max} = 6$	38,260,781	559,175,214,535

Table 1: Nuclei, model space truncations, basis space dimensions, and the number of nonzero matrix elements in the lower-triangle of the sparse symmetric A-body Hamiltonian matrix. The first four lines are with NN forces only, the last with NN and NNN forces.

4.1. Speedup with number of MPI processors

In Fig. 3 we show the speedup over a large range of processor counts, using MPI only, with one MPI process per core on Franklin. The smallest problem, ^{12}C at $N_{\max} = 4$, fits on a single compute node (it needs about 4 GB of memory). Going from a single processor to 28 processors gives a speedup of a factor of 18, and at 100 processors we have a speedup of about 50. Increasing the processor count even further results in rather poor scaling.

However, we have to keep in mind that this is a rather small problem: the total run-time, including reading of the input, the setup phase, and writing of all output, is less than 60 seconds at 120 processors and above. Furthermore, because of the 2-dimensional distribution of the lower-triangle of the matrix, the code is not load-balanced at small

processor counts: The workload of the off-diagonal processors is approximately twice that of the diagonal processors. So on 6 processors, we have 3 "diagonal" and 3 "off-diagonal" processors, the code is not load-balanced at all. However, as we go to (tens of) thousands of processors, this is no longer an issue. In fact, we turn this load-unbalance to our advantage: most of the IO is done by the diagonal processors only, and some of it can overlap calculations on the off-diagonal processors.

Indeed, as we move to a larger problem, see the right panel of Fig. 3, we see reasonable scaling on a range of 500 to 4,000 processors. In particular the construction of the sparse matrix scales quite well, whereas the Lanczos iterations shows a flattening of once we get above about 3,000 processors. As noted above, each Lanczos iteration involves a signifacant amount of global MPI communication; at 3,000 processors each iteration takes only about 1 second. So as we increase the number of MPI processors even further, we will spend more and more time in MPI communication, and less on actual computation.

4.2. Hybrid MPI/OpenMPI

As the model space grows, so does the amount of common data, that each processor has access to. Since available memory is our biggest bottleneck, it would be advantageous if we could eliminate the need for each processor having its own copy of common data. In addition, for very large numbers of MPI processors, the amount of communication, and the number of buffers needed for this communication, becomes more demanding. With current multi-core systems, a hybrid MPI and OpenMP approach might very well be more efficient than pure MPI.

As to the common data, all processors have to access the entire input K-body interaction. For NN interactions, this is generally not a big problem: for the largest model space considered here, ^{6}Li at $N_{\max} = 12$, the total memory needed for the interaction data is about 150 MB. However, the 3-body interaction data require much more memory: for $N_{\max} = 6$, the NNN input interaction file is about 3 GB, and for $N_{\max} = 8$, it is about 33 GB. Each processor has to have access to these input data, and because of our round-robin distribution of the many-body basis states, this is effectively random access. The way we currently deal with this problem is by reading the input file on the n diagonal processors, using MPI-IO and overlapping with computation on the off-diagonal processors. Subsequently, we make several passes in which different subsets of the n diagonal processors broadcast their part of the input data to all processors. In this way all processors have access to all input data, though at every pass we have to re-compute the same quantities. Using OpenMP within a compute node, and MPI between compute nodes, we can significantly reduce the number of passes needed to process the entire input file on each MPI processor.

We have implemented a hybrid MPI/OpenMP code using OpenMP directives in the three most compute-intensive phases of the code (the setup phase is not multithreaded):

- Construction of the matrix
  For the multithreaded determination of the sparsity structure, the number of coarse column groups in our multilevel blocking scheme are distributed over the available threads. The actual calculation of the nonzero matrix elements is parallelized using an OpenMP DO directive for the loop over the columns, which is the outer loop.

- Lanczos iterations
  The sparse matrix-vector multiplication is parallelized using an OpenMP DO directive for the loop over the columns. However, since each matrix block acts both as matrix block for the lower triangle and as the transpose matrix block for the upper triangle, each thread has to have its own private output vectors for the result of the transpose matrix-vector multiplication, which are added inside a OpenMP CRITICAL region. This reduces the anticipated memory gain, in particular when using a large number of threads. The orthogonalization of the output vector against all previous Lanczos vectors is also parallelized with an OpenMP DO directive.

- Evaluation of observables
  The evaluation of select physical observables is parallelized using an OpenMP DO directive for the loop over the columns, just like the calculation of the matrix elements.

In Fig. 4 we show the speedup for the same two cases as shown in Fig. 3. The left panel is on a single compute node on Franklin, whereas the right panel is on Jaguar, with one MPI process per compute node, containing two hex-core processors, so that we can utilize up to 12 threads. On the single processor we see reasonable speedup going from 1 thread to 4 threads, more or less the same for all phases of the code. However, the right panel shows a difference

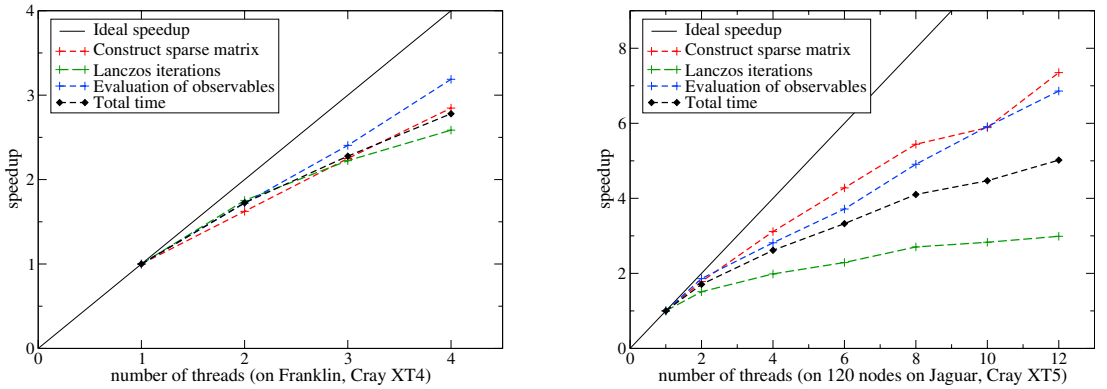

Figure 4: Speedup with number of threads for ^{12}C at $N_{\max} = 4$ on a single node on Franklin (Cray XT4 at NERSC with 8 GB quad-core processors) and for ^{6}Li at $N_{\max} = 12$ (right) on 120 compute nodes on Jaguar (Cray XT5 with two 8 GB hex-core processors per compute node at ORNL), using NN forces only.

between the Lanczos iterations on one hand, and the construction of the matrix and calculation of the observables on the other hand: The speedup for the Lanczos iterations is rather poor. This can be understood if one realizes that each iteration involves a significant amount of MPI communication. Multithreading speeds up the local computations on each node, but not the MPI communication between the nodes. Nevertheless, the performance improves all the way to the maximum of 12 threads, spread over 2 processors per compute node. In other words, if we do need to restrict ourselves to only one MPI process per compute node (i.e. for memory reasons), it is advantageous to utilize all 12 threads on Jaguar.

In order to judge whether it is more efficient to have one MPI process per core, or one MPI process per node with multiple threads, we show in Fig. 5 the total CPU time, that is the actual wallclock time times the number of cores, for two different cases. Ideal scaling would result in a horizontal line; from an application scientist' point of view, it is this total CPU time that we want to minimize. In the left panel we see that for the case of ^{6}Li at $N_{\max} = 12$ it is most efficient to use one MPI process per core, rather than one MPI process per node in combination with OpenMP. However, hybrid MPI/OpenMP version with 4 threads per processor scales somewhat better than the pure MPI case: the solid curves are less steep than the corresponding dotted curves. As the number of cores increases, hybrid MPI/OpenMP will become more efficient than pure MPI.

Indeed, for a larger problem, ^{14}N at $N_{\max} = 8$ which needs at least 10 TB of memory just for storing the matrix and the Lanczos vectors, multithreading is significantly more efficient, see the right panel of Fig. 5. The points at 8,128 and 12,090 cores are runs using pure MPI with one MPI process per core on Franklin and Jaguar respectively, and they are 20% to 50% above the corresponding multithreaded data points. It is interesting to note that the construction of the matrix and evaluation of the observables is more efficient on Jaguar, whereas the Lanczos iterations are more efficient on Franklin. Most likely, the larger L3 cache and greater memory bandwith of the Istanbul processors in Jaguar compared to the Budapest processors in Franklin leads to the higher efficiency in the construction of the matrix and evaluation of the observables. The significant difference in the performance of the Lanczos iterations may be caused by a bottleneck in the communication from the node into the network, as observed in [24], which compares the performance of Jaguar (XT5) having two processors per node with that of Franklin (XT4) having one processor per node. It could also be caused by saturation of the network links on Jaguar. (Jaguar has almost 6 times as many cores as Franklin, but a similar interconnect (both Seastar2).)

Also, using one MPI process with 12 threads per compute node (containing 2 hex-core processors) seems to be slightly less efficient than one MPI process with 6 threads per NUMA processor on Jaguar. This is probably due to memory contention in the former case. As the size of the matrix increases, and thus the required memory increases, the burden of the MPI communication becomes larger, and it becomes more efficient to use 12 threads per compute node, despite the memory contention. In the left panel of Fig. 6 we show the performance of different parts of the

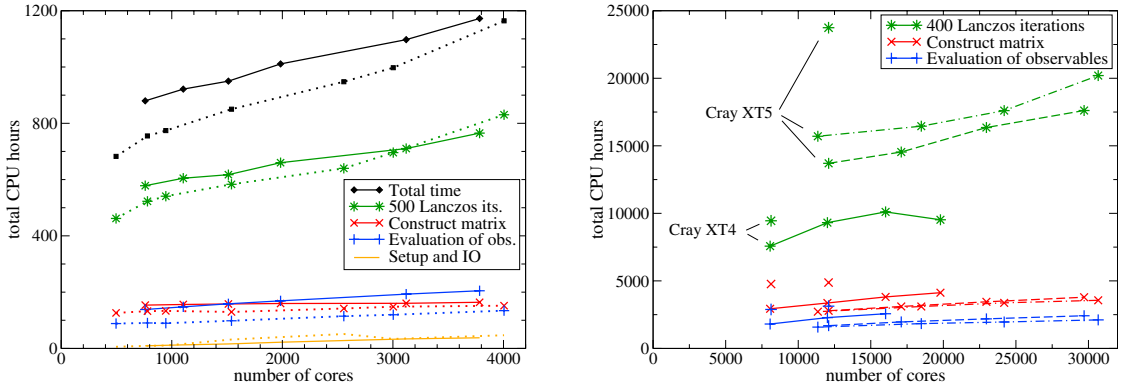

Figure 5: Total CPU time: left for ^{6}Li at $N_{\max} = 12$ on Franklin (Cray XT4 at NERSC) using one MPI process per core (dotted) and one MPI process per node with 4 threads (solid); and right for ^{14}N at $N_{\max} = 8$ using one MPI process per node with 4 threads on Franklin (solid line), and on Jaguar (Cray XT5 at ORNL) using one MPI process per NUMA node with 6 threads (dashed), and one MPI per compute node with 12 threads (dot-dashed), while the single points at 8,128 and 12,090 cores are with one MPI process per core on Franklin and Jaguar respectively.

code on Jaguar before and after the upgrade from quad-core to hex-core for ^{14}Be at $N_{\max} = 8$, which has a dimension of almost 3 billion. Indeed, we see here that for this case, using one MPI process with 12 threads per compute node is slightly more efficient than one MPI process with 6 threads per NUMA processor (the blue vs. green bars).

During September and October 2009 Jaguar was upgraded from dual sockets with quad-core processors to dual sockets with hex-core processors; the memory per node was kept the same (16 GB per compute node, or 8 GB per processor), and the interconnect also remained the same. In addition to the extra cores, the upgrade to the Istanbul processors also meant a slightly higher clock speed (2.3 GHz vs. 2.6 GHz), a significantly larger L3 cache (2 MB vs. 8 MB per processor) and an increased memory bandwidth. Based on the additional cores and increased clock speed, one would expect a speedup of a factor of 1.7 due to this upgrade, while keeping the number of nodes fixed. In practice, the effect of the upgrade is heavily dependent on the workload: the speedup was a factor of 2.13 in the construction of the sparse matrix, whereas the speedup of the Lanczos iterations was only a factor of 1.14. This difference in performance can be explained by the fact that there is a significant amount of MPI communication at each Lanczos iteration across all 7,627 MPI processors, which did not improve with the upgrade. On the other hand, the construction of the matrix benefits greatly from the upgrade: not only the extra cores but also the increased L3 cache and memory bandwith led to a significant speedup of that part of the code. We anticipate that an upgrade of the interconnect from Seastar2 to Gemini would significantly improve the performance of the Lanczos iterations.

Finally, in the right panel of Fig. 6 we plot the total CPU time for ^{13}C at $N_{\max} = 6$ with NN and NNN forces on Franklin with different versions of the code, starting with V10 (blue curve) which was the version at the beginning of our SciDAC grant in early 2007. Over the past 3 years, the overall performance has improved by a factor of 3 to 4, depending on the core count. The current hybrid MPI/OpenMP version is V13, which shows almost perfect scaling from about 3,000 cores to more than 8,000 cores; below 3,000 cores the total CPU time increases because we need multiple passes through the 3-body input data, which decreases the efficiency.

5. Conclusion and Outlook

Our newly developed hybrid MPI/OpenMP CI code for nuclear physics performs well on the current Cray XT4/5 platforms. At large core-counts, typically 5,000 and above, the multithreaded version is significantly more efficient than the pure MPI code, both in terms of memory and in aggregate CPU time. Though we have presented only results obtained on Cray XT4/5 platforms, our conclusions also hold for other multicore systems. The precise number of cores at which the hybrid MPI/OpenMP approach becomes more efficient than the pure MPI approach depends on a number of factors such as the problem size, available memory per processor, memory bandwidth, and network speed.

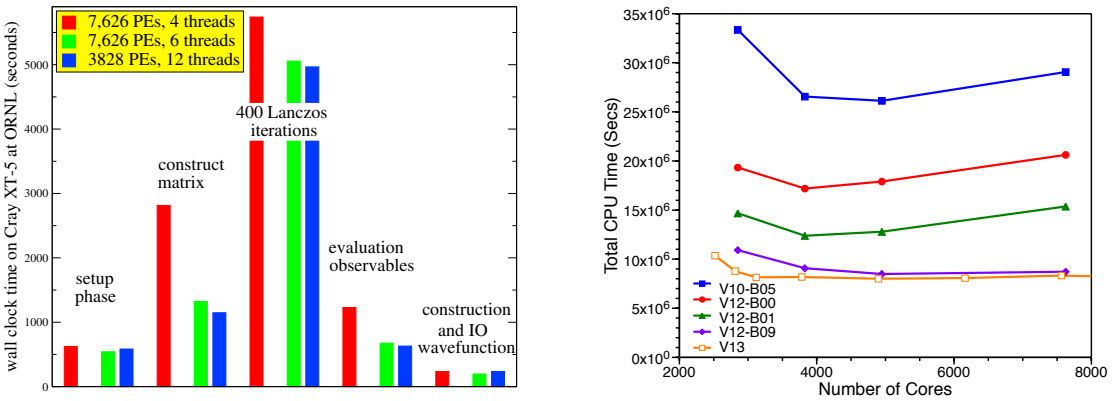

Figure 6: Left: difference in performance on Jaguar (Cray XT5) before (red) and after (green and blue) the upgrade from quad-core to hex-core; Right: Total CPU time for ^{13}C at $N_{\max} = 6$ using NN + NNN forces.

We anticipate to take full advantage of the current and future massively parallel multi-core computer architectures with our hybrid MPI/OpenMP approach.

Acknowledgments

The work was supported in part by Iowa State University under the contract DE-AC02-07CH11358 with the U.S. Department of Energy, by the U.S. Department of Energy under the grants DE-FC02-09ER41582 (UNEDF SciDAC-2) and DE-FG02-87ER40371 (Division of Nuclear Physics) and by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC02-05CH11231. Computational resources were provided by DOE through the National Energy Research Supercomputer Center (NERSC) and through an INCITE award (David Dean, PI).

References

[1] D. R. Entem, R. Machleidt, Accurate nucleon nucleon potential based upon chiral perturbation theory, Phys. Lett. B524 (2002) 93–98. arXiv:nucl-th/0108057, doi:10.1016/S0370-2693(01)01363-6.
[2] D. R. Entem, R. Machleidt, Accurate Charge-Dependent Nucleon-Nucleon Potential at Fourth Order of Chiral Perturbation Theory, Phys. Rev. C68 (2003) 041001. arXiv:nucl-th/0304018, doi:10.1103/PhysRevC.68.041001.
[3] R. B. Wiringa, V. G. J. Stoks, R. Schiavilla, An Accurate nucleon-nucleon potential with charge independence breaking, Phys. Rev. C51 (1995) 38–51. arXiv:nucl-th/9408016, doi:10.1103/PhysRevC.51.38.
[4] J. Carlson, V. R. Pandharipande, R. B. Wiringa, Three-nucleon interaction in 3-, 4-, and ∞-body systems, Nucl. Phys. A401 (1983) 59–85. doi:10.1016/0375-9474(83)90336-6.
[5] S. C. Pieper, V. R. Pandharipande, R. B. Wiringa, J. Carlson, Realistic models of pion-exchange three-nucleon interactions, Phys. Rev. C64 (2001) 014001. arXiv:nucl-th/0102004, doi:10.1103/PhysRevC.64.014001.
[6] A. Negret, et al., Gamow-Teller Strengths in the A=14 Multiplet: A Challenge to the Shell Model, Phys. Rev. Lett. 97 (2006) 062502. doi:10.1103/PhysRevLett.97.062502.
[7] P. Navratil, V. G. Gueorguiev, J. P. Vary, W. E. Ormand, A. Nogga, Structure of A=10-13 nuclei with two- plus three-nucleon interactions from chiral effective field theory, Phys. Rev. Lett. 99 (2007) 042501. arXiv:nucl-th/0701038, doi:10.1103/PhysRevLett.99.042501.
[8] G. Hagen, T. Papenbrock, D. J. Dean, M. Hjorth-Jensen, Medium-mass nuclei from chiral nucleon-nucleon interactions, Phys. Rev. Lett. 101 (2008) 092502. arXiv:0806.3478, doi:10.1103/PhysRevLett.101.092502.
[9] P. Navratil, J. P. Vary, B. R. Barrett, Properties of ^{12}C in the *ab initio* nuclear shell-model, Phys. Rev. Lett. 84 (2000) 5728–5731. arXiv:nucl-th/0004058, doi:10.1103/PhysRevLett.84.5728.
[10] S. K. Bogner, et al., Convergence in the no-core shell model with low-momentum two-nucleon interactions, Nucl. Phys. A801 (2008) 21–42. arXiv:0708.3754, doi:10.1016/j.nuclphysa.2007.12.008.
[11] P. Maris, J. P. Vary, A. M. Shirokov, Ab initio no-core full configuration calculations of light nuclei, Phys. Rev. C79 (2009) 014308. arXiv:0808.3420, doi:10.1103/PhysRevC.79.014308.
[12] P. Maris, A. M. Shirokov, J. P. Vary, Ab initio nuclear structure simulations: the speculative ^{14}F nucleusarXiv:0911.2281.
[13] J. P. Vary, The many-fermion dynamics shell-model code, unpublished (1992).

[14] J. P. Vary, D. C. Zheng, The many-fermion dynamics shell-model code, unpublished (1994).
[15] P. Sternberg, E. G. Ng, C. Yang, P. Maris, J. P. Vary, M. Sosonkina, H. V. Le, Accelerating configuration interaction calculations for nuclear structure, in: SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, IEEE Press, Piscataway, NJ, USA, 2008, pp. 1–12.
[16] S. L. Graham, M. Snir, C. A. Patterson, N. R. C. U. C. on the Future of Supercomputing, Getting up to speed the future of supercomputing, Washington, DC : National Academies Press, 2005, "Committee on the Future of Supercomputing, Computer Science and Telecommunications Board, Division on Engineering and Physical Sciences, National Research Council of the National Academies.".
URL http://www.nap.edu/books/0309095026/html
[17] M. P. I. Forum, Mpi: A message-passing interface standard (1994).
[18] L. Dagum, R. Menon, Openmp: An industry-standard api for shared-memory programming, Computing in Science and Engineering 5 (1998) 46–55. doi:http://doi.ieeecomputersociety.org/10.1109/99.660313.
[19] E. L. Lusk, A. Chan, Early experiments with the openmp/mpi hybrid programming model, in: R. Eigenmann, B. R. de Supinski (Eds.), OpenMP in a New Era of Parallelism, 4th International Workshop, IWOMP 2008, West Lafayette, IN, USA, May 12-14, 2008, Proceedings, 2008, pp. 36–47.
[20] R. Rabenseifner, G. Hager, G. Jost, Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes, in: D. E. Baz, F. Spies, T. Gross (Eds.), Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2009, Weimar, Germany, 18-20 Febuary 200, 2009, pp. 427–436.
[21] M. Sosonkina, A. Sharda, A. Negoita, J. P. Vary, Integration of ab initio nuclear physics calculations with optimization techniques, in: Computational Science - ICCS 2008, 8th International Conference, Kraków, Poland, June 23-25, 2008, Proceedings, Part I, 2008, pp. 833–842.
[22] J. P. Vary, P. Maris, E. Ng, C. Yang, M. Sosonkina, *Ab initio* nuclear structure - the large sparse matrix eigenvalue problem, J. Phys. Conf. Ser. 180 (2009) 012083. arXiv:0907.0209, doi:10.1088/1742-6596/180/1/012083.
[23] N. Laghave, M. Sosonkina, P. Maris, J. P. Vary, Benefits of parallel i/o in ab initio nuclear physics calculations, in: Computational Science - ICCS 2009, 9th International Conference, Baton Rouge, LA, USA, May 25-27, 2009, Proceedings, Part I, 2009, pp. 84–93.
[24] P. H. Worley, R. F. Barrett, J. A. Kuehn, Early evaluation of the cray xt5, in: Proc. 51st Cray User Group Conference, 2009.